

Introduction to Gunk

The Modern Toolkit for Building Microservices

Jason Wangsadinata

Systems Developer @ Brankas

First and foremost...

- ▶ Thank you to:
 - ▶ GoJakarta + ScaleJakarta for organizing.
 - ▶ Xendit office for hosting.
 - ▶ All of you guys for attending.

A little bit about myself

◎ Jason Wangsadinata

- @jwangsadinata
- Systems Developer @ Brankas
- Passionate about building scalable services and finding best practices to do so.



**How do we make
computers talk to each
other?**

How do we make
applications that can
talk to each other?

Web APIs

SOAP

REST

HTTP + JSON

(REST is not the above btw)

Why HTTP/REST?

- ▶ It is easy to understand (*text protocol*)
- ▶ Web infrastructure is already built on top of HTTP
- ▶ Great tooling for testing, inspection, modification
- ▶ Loose coupling between client/server makes changes relatively easy
- ▶ High-quality http implementations in every language

Why HTTP/REST - continued...

▶ JSON

- ▶ It is simple
- ▶ Looks like plain JavaScript object / Python dictionaries
- ▶ XML was a little bit too verbose
- ▶ Movement away from SOAP(xml) towards REST

**Looks like REST
is perfect, bye...**

Not quite...

- ▶ **JSON + HTTP is nice but it is not a silver bullet**
- ▶ It is not the best when:
 - ▶ **Performance matters**
 - ▶ Readability does not really matter
 - ▶ **Type safety is required**
- ▶ Standard contract between applications/computers are needed (continued next page...)

Why REST API is not so good?

- ▶ **No formal (machine-readable) API contract**
 - ▶ Writing client libraries requires humans
 - ▶ **Humans (myself included) hate writing client libraries**
- ▶ Streaming is difficult
- ▶ Bi-directional streaming is not possible at all
- ▶ Inefficient (text representation are not efficient for networks)
- ▶ Hard to get many resources in a single request (think GraphQL)

“If I never write another REST client library in my life **I will die happy**”

- Alan Shreve, ngrok

Protocol Buffers

- ▶ Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data
- ▶ Google's Interface Description Language (IDL)
- ▶ Think XML, but smaller, faster, and simpler
- ▶ Has data types like `message`, `enum` and `service`.
- ▶ Language and platform neutral

GRPC: What is it?

- ▶ A high performance, open-source, universal RPC framework
- ▶ It stands for **g**RPC **R**emote **P**rocedure **C**alls
- ▶ Part of Cloud Native Computing Foundation (cncf.io)
- ▶ Open-source version of *Stubby* used in *Google*.



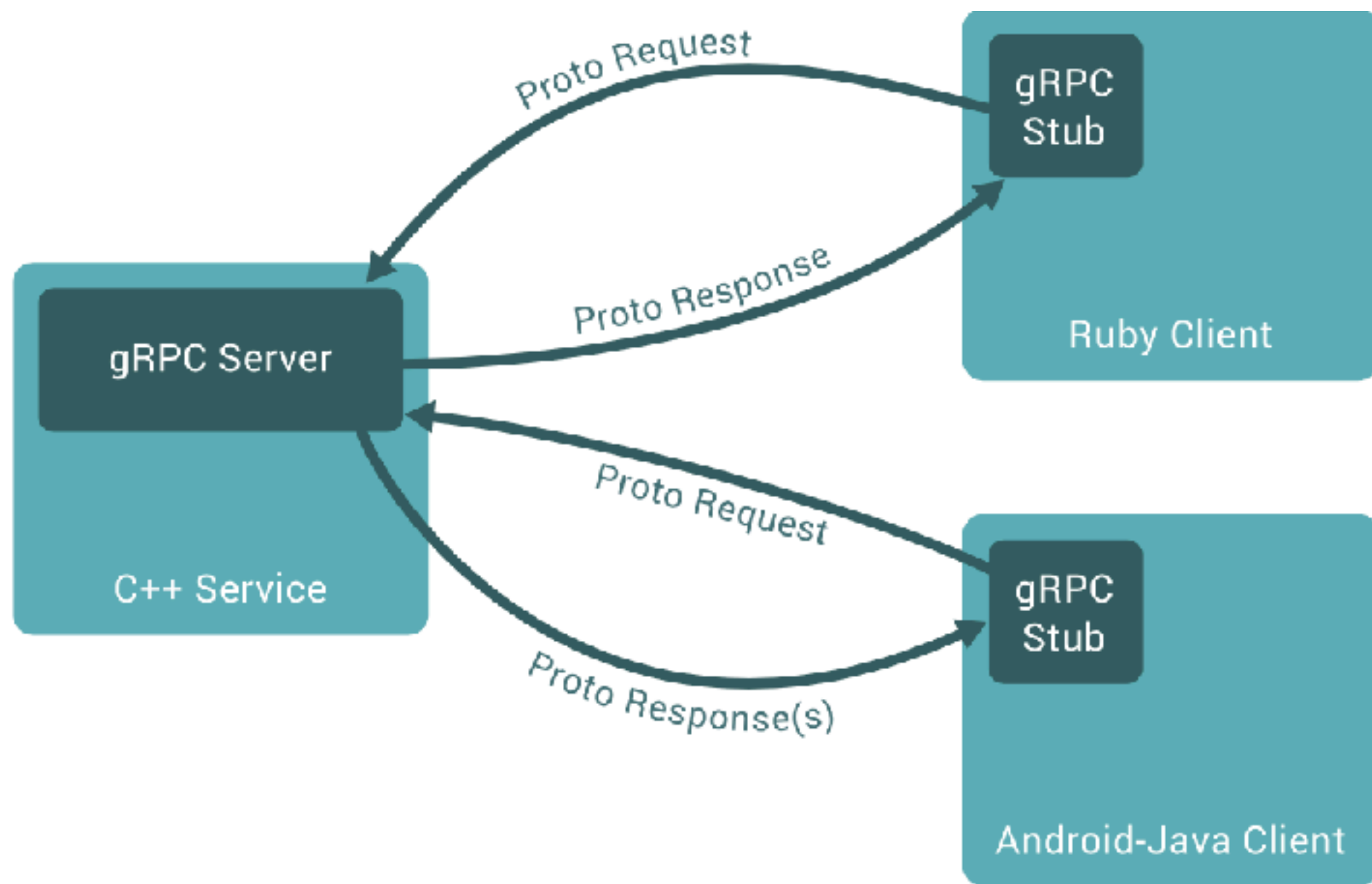
gRPC — continued

- ▶ Abstractions and best practices on how to design RPCs
- ▶ Default implementation(s) from Google
- ▶ Extension points to plug custom implementations and modifications
- ▶ Supports 10+ programming languages
- ▶ In particular, Go has first class support for *protobuf* and *gRPC*.



GRPC: On the wire

- ▶ HTTP/2
- ▶ protobuf serialization (pluggable)
- ▶ Clients open one long-lived connection to a GRPC server
 - ▶ A new HTTP/2 stream for each RPC call
 - ▶ Allows simultaneous in-flight RPC calls
- ▶ Allows client-side and server-side streaming
- ▶ Built on:
 - ▶ HTTP/2, IDL, protobufs



GRPC: Implementation

- ▶ Three high-performance event loop driven implementations
- ▶ **Go**
 - ▶ Pure Go implementation using Go stdlib `crypto/tls` package
- ▶ **C**
 - ▶ Ruby, Python, Node.js, PHP, Objective-C, C++, C# are all bindings to the `C Core`
 - ▶ PHP via PECL extension (apache or nginx/php-fpm)
- ▶ **Java**
 - ▶ Netty + BoringSSL via JNI

**enough background, tell me
about gunk**

Gunk

- ▶ (<https://github.com/gunk/gunk>)
- ▶ Wrapper around the protobuf's `protoc` compiler.
- ▶ It stands for "**G**unk **U**nified **N**-terface **K**ompiler".
- ▶ Written in simple and idiomatic Go.
- ▶ Intuitive to use.

Why build Gunk?

- ▶ Create an Go-compatible way of defining definitions that can be read and handled by `go/*` package.
- ▶ Less overhead in compiling the API definitions.
- ▶ Write an intuitive tool to quickly generate and build services.

Gunk file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name    string `pb:"1" json:"name"`
    ID      int    `pb:"2" json:"id"`
    Rarity  Rarity `pb:"3" json:"rarity"`
}
```

Gunk file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name  string `pb:"1" json:"name"`
    ID    int    `pb:"2" json:"id"`
    Rarity Rarity `pb:"3" json:"rarity"`
}
```

Looks like a regular .go file to me?



Gunk file vs protobuf file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name    string `pb:"1" json:"name"`
    ID      int    `pb:"2" json:"id"`
    Rarity  Rarity `pb:"3" json:"rarity"`
}
```

```
syntax = "proto3";
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

Gunk file vs protobuf file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name string `pb:"1" json:"name"`
    ID    int    `pb:"2" json:"id"`
    Rarity Rarity `pb:"3" json:"rarity"`
}
```

```
syntax = "proto3";
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

Gunk file vs protobuf file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name  string `pb:"1" json:"name"`
    ID    int    `pb:"2" json:"id"`
    Rarity Rarity `pb:"3" json:"rarity"`
}
```

```
syntax = "proto3";
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

Gunk file vs protobuf file

```
package pokemon
```

```
type PokemonService interface {  
    Get(GetReq) Pokemon  
}
```

```
type GetReq struct {  
    ID int `pb:"1" json:"id"`  
}
```

```
type Rarity int
```

```
const (  
    NORMAL Rarity = iota  
    LEGENDARY  
    MYTHIC  
)
```

```
type Pokemon struct {  
    Name string `pb:"1" json:"name"`  
    ID int `pb:"2" json:"id"`  
    Rarity Rarity `pb:"3" json:"rarity"`  
}
```

```
syntax = "proto3";
```

```
package pokemon;
```

```
service PokemonService {  
    rpc Get (GetReq) returns (Pokemon) {}  
}
```

```
message GetReq {  
    int32 id = 1;  
}
```

```
message Pokemon {  
    string name = 1;  
    int32 id = 2;
```

```
    enum Rarity {  
        NORMAL = 0;  
        LEGENDARY = 1;  
        MYTHIC = 2;  
    }
```

```
    Rarity rarity = 3;  
}
```

Gunk file vs protobuf file

```
package pokemon

type PokemonService interface {
    Get(GetReq) Pokemon
}

type GetReq struct {
    ID int `pb:"1" json:"id"`
}

type Rarity int

const (
    NORMAL Rarity = iota
    LEGENDARY
    MYTHIC
)

type Pokemon struct {
    Name  string `pb:"1" json:"name"`
    ID    int    `pb:"2" json:"id"`
    Rarity Rarity `pb:"3" json:"rarity"`
}
```

```
syntax = "proto3";
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

Usage

```
$ gunk
```

```
usage: gunk [<flags>] <command> [<args> ...]
```

Gunk Unified N-terface Kompiler command-line tool.

Flags:

-h, --help Show context-sensitive help (also try --help-long and --help-man).

Commands:

help [<command>...]

Show help.

generate [<flags>] [<patterns>...]

Generate code from Gunk packages.

convert [<flags>] [<file>]

Convert Proto file to Gunk file.

format [<patterns>...]

Format Gunk code.

Usage

\$ gunk

usage: gunk [<flags>] <command> [<args> ...]

Gunk Unified N-terface Kompiler command-line tool.

Flags:

-h, --help Show context-sensitive help (also try --help-long and --help-man).

Commands:

help [<command>...]

Show help.

generate [<flags>] [<patterns>...]

Generate code from Gunk packages.

convert [<flags>] [<file>]

Convert Proto file to Gunk file.

format [<patterns>...]

Format Gunk code.

Usage

```
$ gunk
```

```
usage: gunk [<flags>] <command> [<args> ...]
```

Gunk Unified N-terface Kompiler command-line tool.

Flags:

-h, --help Show context-sensitive help (also try --help-long and --help-man).

Commands:

```
help [<command>...]
```

Show help.

```
generate [<flags>] [<patterns>...]
```

Generate code from Gunk packages.

```
convert [<flags>] [<file>]
```

Convert Proto file to Gunk file.

```
format [<patterns>...]
```

Format Gunk code.

Usage

```
$ gunk
```

```
usage: gunk [<flags>] <command> [<args> ...]
```

Gunk Unified N-terface Kompiler command-line tool.

Flags:

-h, --help Show context-sensitive help (also try --help-long and --help-man).

Commands:

```
help [<command>...]
```

Show help.

```
generate [<flags>] [<patterns>...]
```

Generate code from Gunk packages.

```
convert [<flags>] [<file>]
```

Convert Proto file to Gunk file.

```
format [<patterns>...]
```

Format Gunk code.

enuf talk, let's gunk

Write the server

```
package main

// imports are truncated on this slide for clarity

type Server struct{}

func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {
    // TODO: write the service implementation here
}

func main() {
    l, err := net.Listen("tcp", ":9090")
    if err != nil {
        log.Fatal(err)
    }
    s := grpc.NewServer()
    reflection.Register(s)
    pb.RegisterPokemonServiceServer(s, &Server{})
    log.Fatal(s.Serve(l))
}
```

Write the server

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
type Server struct{}
```

```
func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {
```

```
    // TODO: write the service implementation here
```

-> write the server logic here

```
}
```

```
func main() {
```

```
    l, err := net.Listen("tcp", ":9090")
```

```
    if err != nil {
```

```
        log.Fatal(err)
```

```
    }
```

```
    s := grpc.NewServer()
```

```
    reflection.Register(s)
```

```
    pb.RegisterPokemonServiceServer(s, &Server{})
```

```
    log.Fatal(s.Serve(l))
```

```
}
```

Write the server

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
type Server struct{}
```

```
func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {  
    // TODO: write the service implementation here  
}
```

```
func main() {  
    l, err := net.Listen("tcp", ":9090")  
    if err != nil {  
        log.Fatal(err)  
    }  
    s := grpc.NewServer()  
    reflection.Register(s)  
    pb.RegisterPokemonServiceServer(s, &Server{})  
    log.Fatal(s.Serve(l))  
}
```

-> bind to a port for serving

Write the server

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
type Server struct{}
```

```
func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {  
    // TODO: write the service implementation here  
}
```

```
func main() {  
    l, err := net.Listen("tcp", ":9090")  
    if err != nil {  
        log.Fatal(err)  
    }  
    s := grpc.NewServer()  
    reflection.Register(s)  
    pb.RegisterPokemonServiceServer(s, &Server{})  
    log.Fatal(s.Serve(l))  
}
```

-> initialize a new server

Write the server

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
type Server struct{}
```

```
func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {  
    // TODO: write the service implementation here  
}
```

```
func main() {  
    l, err := net.Listen("tcp", ":9090")  
    if err != nil {  
        log.Fatal(err)  
    }  
    s := grpc.NewServer()  
    reflection.Register(s)  
    pb.RegisterPokemonServiceServer(s, &Server{})  
    log.Fatal(s.Serve(l))  
}
```

-> enable reflection service

Write the server

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
type Server struct{}
```

```
func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {  
    // TODO: write the service implementation here  
}
```

```
func main() {  
    l, err := net.Listen("tcp", ":9090")  
    if err != nil {  
        log.Fatal(err)  
    }  
    s := grpc.NewServer()  
    reflection.Register(s)  
    pb.RegisterPokemonServiceServer(s, &Server{})  
    log.Fatal(s.Serve(l))  
}
```

-> register service to our server

Write the server

```
package main

// imports are truncated on this slide for clarity

type Server struct{}

func (s *Server) Get(ctx context.Context, req *pb.GetReq) (*pb.Pokemon, error) {
    // TODO: write the service implementation here
}

func main() {
    l, err := net.Listen("tcp", ":9090")
    if err != nil {
        log.Fatal(err)
    }
    s := grpc.NewServer()
    reflection.Register(s)
    pb.RegisterPokemonServiceServer(s, &Server{})
    log.Fatal(s.Serve(l))
}
```

-> start serving

Write the client

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
func main() {  
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())  
    if err != nil {  
        log.Fatalf("did not connect: %v", err)  
    }  
    defer conn.Close()  
    c := pb.NewPokemonServiceClient(conn)  
  
    ctx, cancel := context.WithTimeout(context.Background(), time.Second)  
    defer cancel()  
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})  
    if err != nil {  
        log.Fatalf("could not get: %v", err)  
    }  
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String())  
}
```

Write the client

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
func main() {
```

```
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())
```

-> connect to the server

```
    if err != nil {
```

```
        log.Fatalf("did not connect: %v", err)
```

```
    }
```

```
    defer conn.Close()
```

```
    c := pb.NewPokemonServiceClient(conn)
```

```
    ctx, cancel := context.WithTimeout(context.Background(), time.Second)
```

```
    defer cancel()
```

```
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})
```

```
    if err != nil {
```

```
        log.Fatalf("could not get: %v", err)
```

```
    }
```

```
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String())
```

```
}
```

Write the client

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
func main() {  
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())  
    if err != nil {  
        log.Fatalf("did not connect: %v", err)  
    }  
    defer conn.Close()  
    c := pb.NewPokemonServiceClient(conn)  
  
    ctx, cancel := context.WithTimeout(context.Background(), time.Second)  
    defer cancel()  
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})  
    if err != nil {  
        log.Fatalf("could not get: %v", err)  
    }  
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String())  
}
```

-> init the client stub

Write the client

```
package main

// imports are truncated on this slide for clarity

func main() {
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    c := pb.NewPokemonServiceClient(conn)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second) -> set context
    defer cancel()
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})
    if err != nil {
        log.Fatalf("could not get: %v", err)
    }
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String())
}
```

Write the client

```
package main
```

```
// imports are truncated on this slide for clarity
```

```
func main() {  
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())  
    if err != nil {  
        log.Fatalf("did not connect: %v", err)  
    }  
    defer conn.Close()  
    c := pb.NewPokemonServiceClient(conn)  
  
    ctx, cancel := context.WithTimeout(context.Background(), time.Second)  
    defer cancel()  
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})  
    if err != nil {  
        log.Fatalf("could not get: %v", err)  
    }  
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String())  
}
```

-> call the client stub

Write the client

```
package main

// imports are truncated on this slide for clarity

func main() {
    conn, err := grpc.Dial("localhost:9090", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    c := pb.NewPokemonServiceClient(conn)

    ctx, cancel := context.WithTimeout(context.Background(), time.Second)
    defer cancel()
    r, err := c.Get(ctx, &pb.GetReq{ID: int32(25)})
    if err != nil {
        log.Fatalf("could not get: %v", err)
    }
    log.Printf("Pokemon #%d is %s (%s)", r.ID, r.Name, r.Rarity.String()) -> use the response
}
```

Verify that the service works

- ▶ Start the server
 - ▶ `$ cd server`
 - ▶ `$ go build && ./server`
- ▶ In another window, start the client
 - ▶ `$ cd client`
 - ▶ `$ go build && ./client`
- ▶ Verify that the client received the response

demo time

State and future of gunk

- ▶ Currently under active development.
- ▶ Enable support for all major languages and plugins.
- ▶ Allow support for grpc streams.
- ▶ Ramp up on project documentation.

Contributing

- ▶ Try it! Best way to contribute is to actually use the project.
- ▶ Document any issues you encounter using the issue tracker.
- ▶ Features you want to see? Also raise it via the issue tracker.
- ▶ Know a better implementation/solution, send PR our way.

Shameless plug

- ▶ A copy of this slide is available on my GitHub (<https://github.com/jwangsadinata/talks/intro-to-gunk>)
- ▶ If you like these kinds of stuff, check out Brankas (<https://brank.as>).
- ▶ Definitely join the meetup group for GoJakarta and ScaleJakarta
 - ▶ **GoJakarta** (<https://www.meetup.com/GoJakarta/>)
 - ▶ **ScaleJakarta** (<https://www.meetup.com/ScaleJakarta/>)
- ▶ I recently wrote a Kubernetes utility called k8shhh, check it out and give it some love:
 - ▶ k8shhh (<https://github.com/jwangsadinata/k8shhh>)
- ▶ Shoutout to Jonathan Pentecost and Daniel Martì for their contributions and inputs.

questions?

terima kasih